US005375074A

United States Patent [19]

Greenberg et al.

[11] Patent Number: 5,375,074

[45] Date of Patent: Dec. 20, 1994

[54] **UNBOUNDEDLY PARALLEL SIMULATIONS**

[75] Inventors: **Albert G. Greenberg**, Millburn; **Boris D. Lubachevsky**, Bridgewater, both of N.J.; **Israel Mitrani**, Newcastle-upon-Tyne, United Kingdom

[73] Assignee: **AT&T Corp.**, Murray Hill, N.J.

[21] Appl. No.: **468,524**

[22] Filed: **Jan. 23, 1990**

[51] Int. Cl.[5] **G06F 15/60**

[52] U.S. Cl. **364/578**

[58] Field of Search ..... 364/578, 976.5, 916.3, 364/931.41; 371/23; 340/286 M, 515

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,306,286	12/1981	Cocke et al.	364/200
4,751,637	6/1988	Catlin	364/200
4,814,978	3/1989	Dennis	364/900
4,866,605	9/1989	Nakano et al.	364/200
4,901,260	2/1990	Lubachevsky	364/578
4,914,612	4/1990	Beece et al.	364/578
4,930,102	5/1990	Jennings	364/900
4,942,615	7/1990	Hirose	364/578

OTHER PUBLICATIONS

Margolus; "Cellular-Automatic Supercomputers for Fluid-Dynamics Modelling" *Physical Review Letters vol. 56, 1986.*

Ulrich; "Serial/Parallel Event Scheduling for the Simulation of Large Systems"; Acm 1968.

Clouqueur et al; "RAPL, a Cellular Automation Machine for Fluid Dynamics"; Complex Systems 1987.

Chandy and Sherman; "Space-Time and simulation", Proceedings of Distributed Simulation 1989, Conference of the Society for Computer Simulation.

Batcher; "Sorting networks and their applications," AFIPS SJCC 32, 1968.

Primary Examiner—Ellis B. Ramirez

Attorney, Agent, or Firm—Henry T. Brendzel

[57] **ABSTRACT**

Efficient simulation is achieved by employing a highly efficient ordering of the events to be simulated. Specifically, the events to be simulated are grouped into layers and the layers are simulated in order. Each of the layers consists of events that are either strictly independent of the other events in the layer or are dependent of other events in the layer but possess a particular attribute. That attribute is one that permits the use of an associative operator. This operator allows the simulation of N events in O(log N) computation iterations.

33 Claims, 5 Drawing Sheets

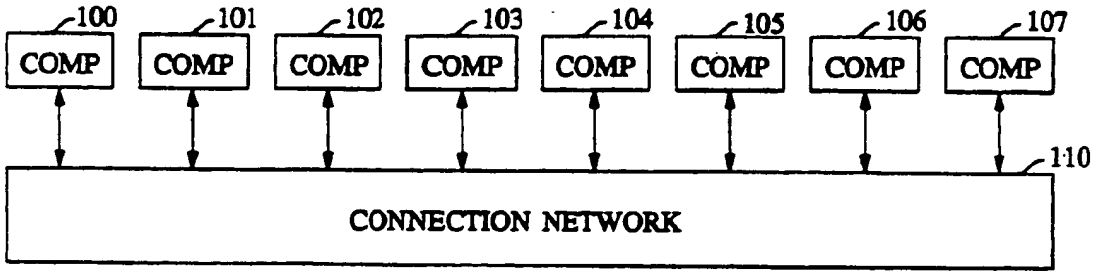


FIG. 1

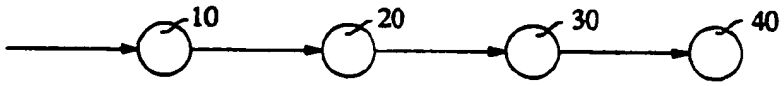


FIG. 3

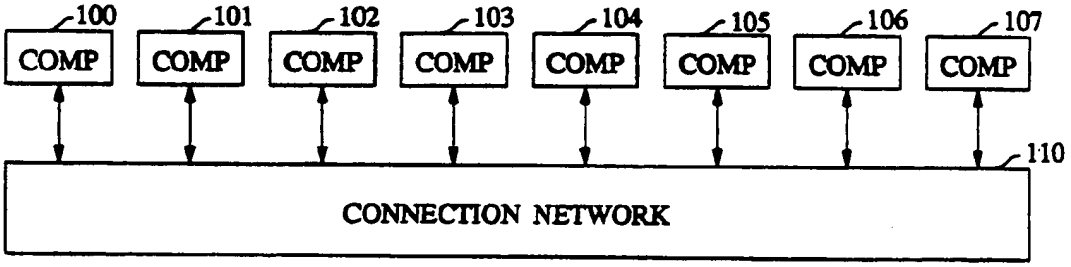


FIG. 4

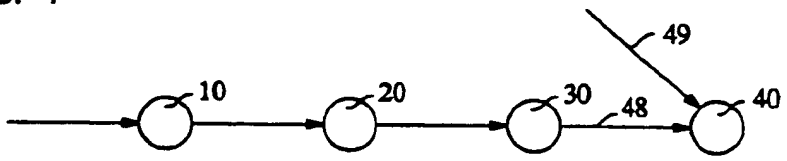


FIG. 5

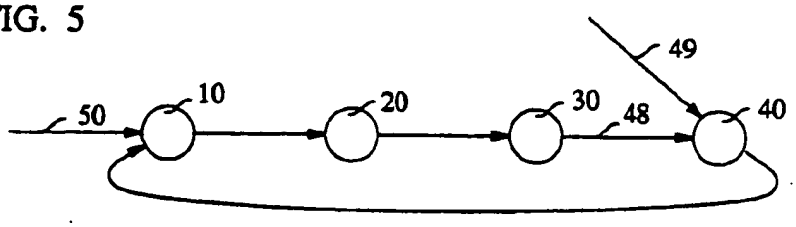


FIG. 2

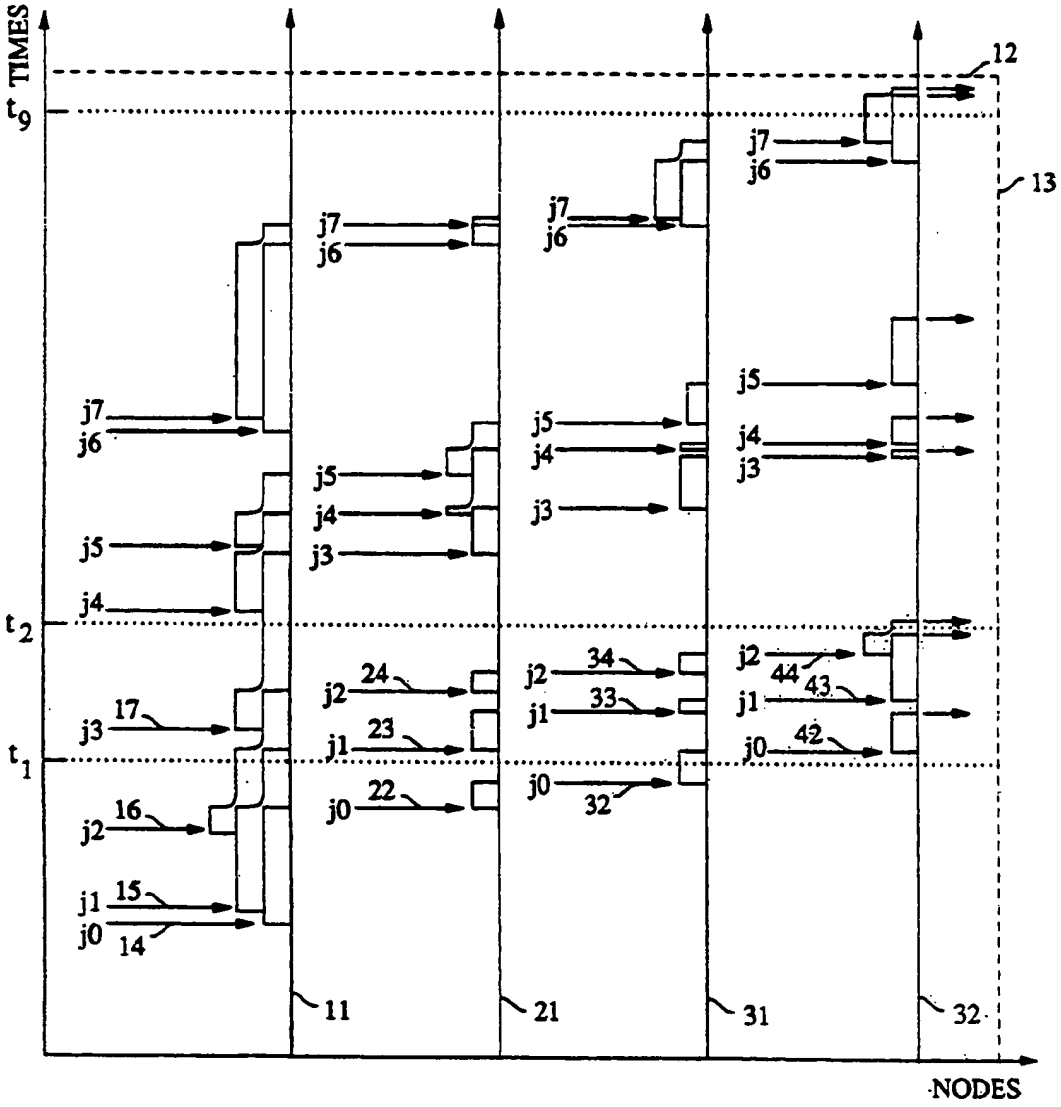

TIMES
t9
t2
t1
12
13
j7
j6
j5
j4
j3
j2
j1
j0
17
16
15
14
24
23
22
34
33
32
44
43
42
11
21
31
32
NODES

FIG. 6

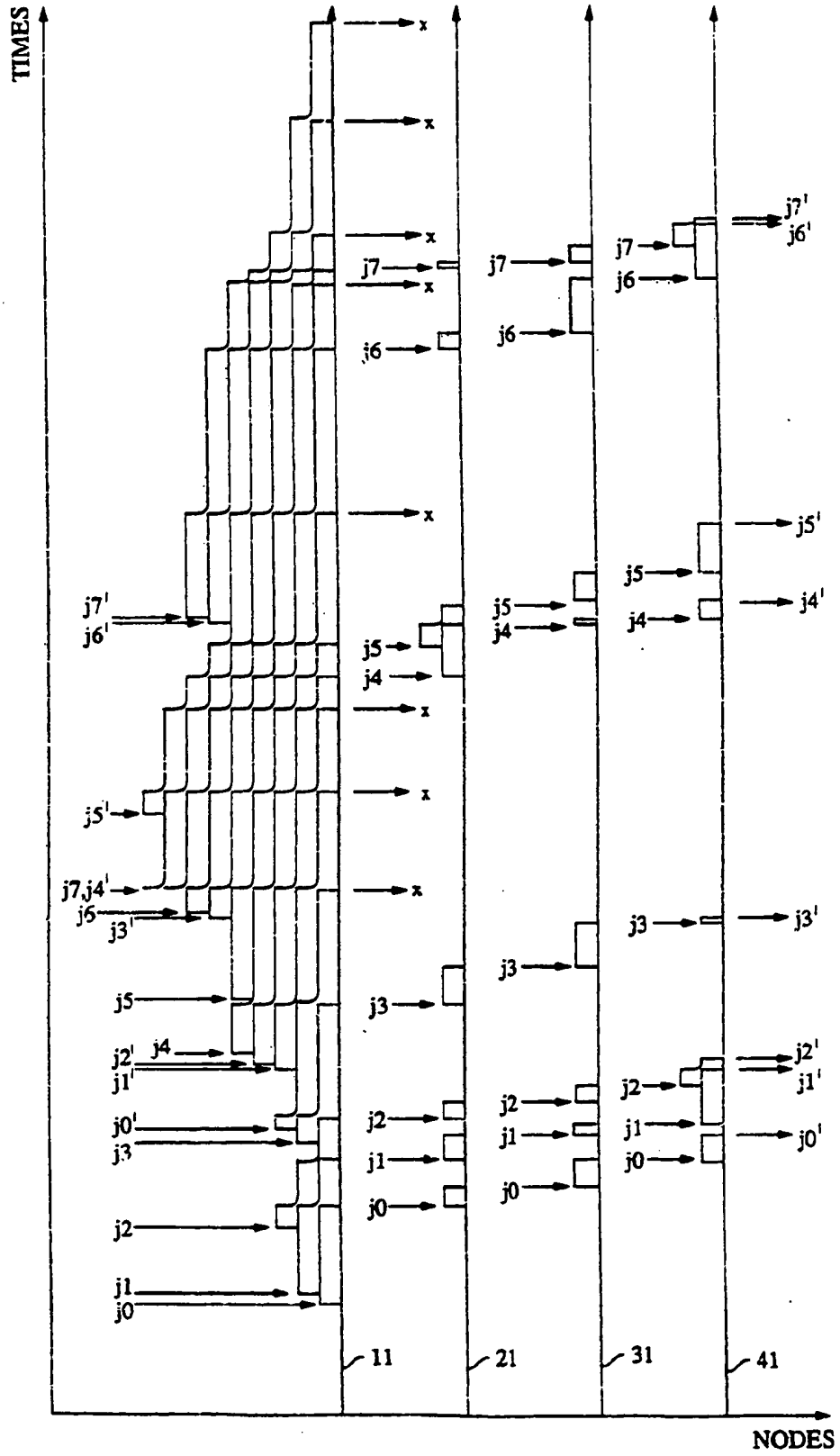

TIMES
NODES
11
21
31
41
x
j7
j6
j5
j4
j3
j2
j1
j0
j7'
j6'
j5'
j4'
j3'
j2'
j1'
j0'

FIG. 7

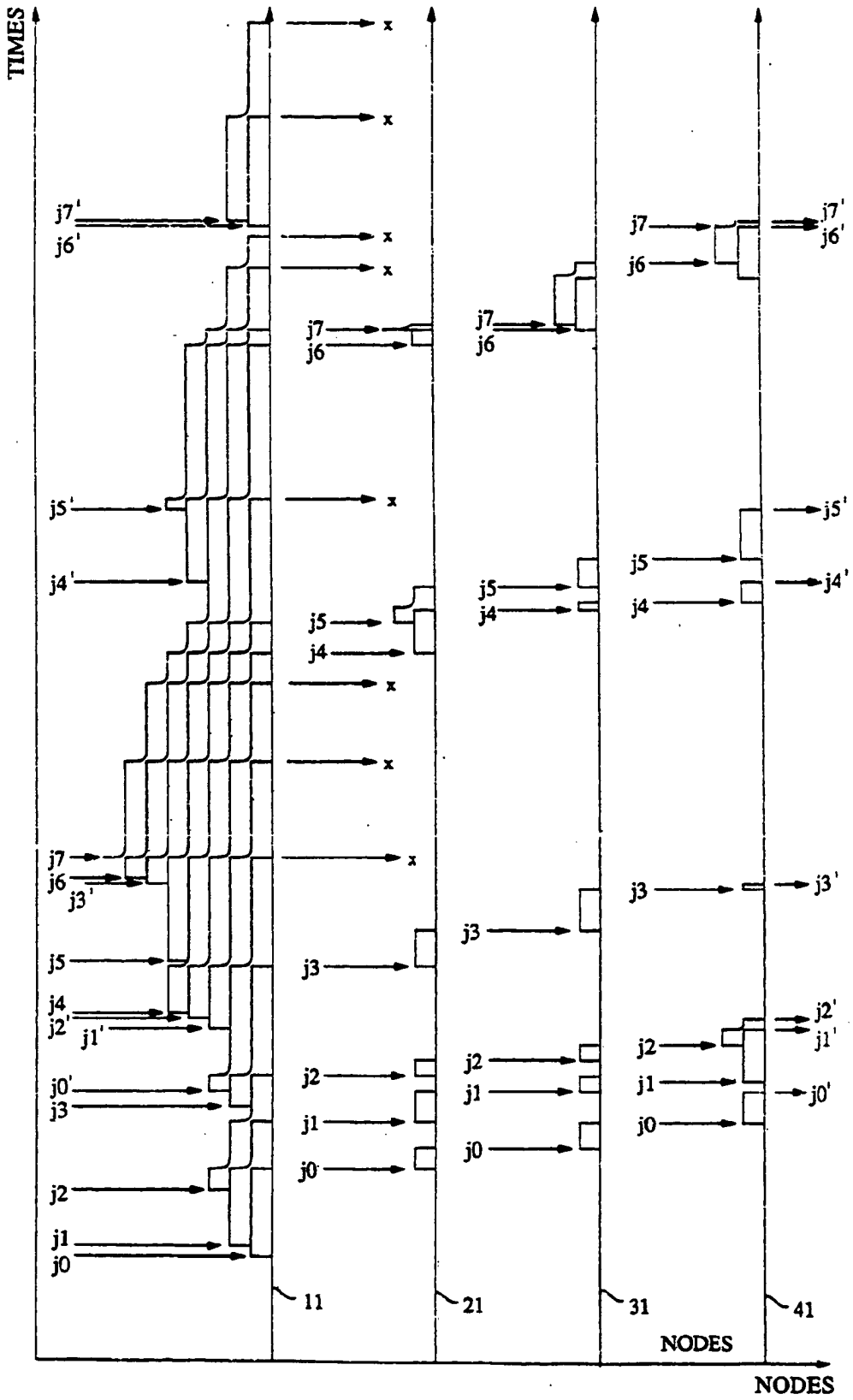
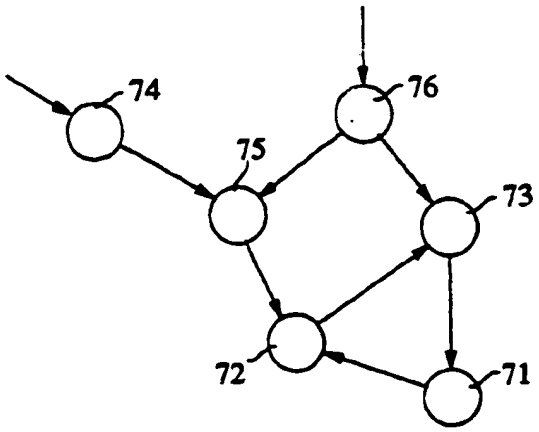

TIMES
x
x
x
x
x
x
x
x
x
j7'
j6'
j7
j6
j5'
j4'
j5
j4
j7
j6
j3'
j5
j4
j2'
j1'
j0'
j3
j2
j1
j0
j3
j2
j1
j0
j7
j6
j5
j4
j3
j2
j1
j0
j7'
j6'
j5'
j4'
j3'
j2'
j1'
j0'
11
21
31
41
NODES
NODES

FIG. 8



UNBOUNDEDLY PARALLEL SIMULATIONS

BACKGROUND OF THE INVENTION

This invention relates to discrete events simulation and, more specifically, to efficient simulation of events in a multiprocessor environment.

Simulation of a discrete event system traditionally entails simulating events in time order, beginning at some initial simulated time and progressing forward in time. This approach normally utilizes a global "simulated time" clock and an event list. Formerly, simulations were performed on a single computer and events were simulated seriatim, progressing forward in simulated time. To improve simulation speed, it was natural to look at the use of more than one processor to simulate the event list and, indeed, some speed improvement was realized by employing more processors even though the basic approach remained the same. However, the improvement in simulation time came at the expense of a communication burden. The processors that did the simulating had to communicate at least some of the simulation results to other processors, and the communication burden grew very quickly as the number of processors increased.

Since the physical system that is simulated typically comprises a number of "nodes" in which events occur (the term "nodes" herein intends to encompass objects, stations, locations, etc. that are associated with events), the obvious division of labor among the processors was to assign a group of nodes to each of the cooperating processors. Restricting each processor to simulate events that occur at certain nodes, however, imposed a synchronization requirement. A processor could not be allowed to simulate events of a certain node until it was known that no node at any of the other processors would send a message that would affect the simulation of that certain node (if "back-tracking" was not to be employed). The high communication burden between the computers sparked interest in simulation algorithms that would reduce this burden. One such algorithm is described, for example, in U.S. patent application Ser. No. 07/114369, titled "Bounded Lag Distributed Discrete Event Simulation Method and Apparatus" and filed on Oct. 28, 1987, M.S. Pat. No. 4,901,260.

A recent publication by Chandy and Sherman ("Space-Time and simulation", *Proceedings of Distributed Simulation* 1989 conference of the Society for Computer Simulation), provides an overview of various simulation techniques applicable to multiprocessor arrangements. It describes an approach where the all of the events of all of the nodes at all times are treated as an a priori known whole. They depict this "whole" as a rectangle where the nodes are marked along the x axis and time is marked along the y axis. The events to be simulated are points within that rectangle or, more specifically, points along vertical time lines that are associated with the nodes.

The concept proposed by Chandy and Sherman is to divide the rectangle into a chosen number of arbitrary regions. The regions may divide the rectangle with vertical cuts and/or with horizontal cuts. The former segregates nodes while the latter segregates time. Each region represents a process that is assigned to a processor. Presumably, more than one process can be assigned to a processor. Still, having preselected the regions and made the assignments, the problem remains to determine which events will occur in which regions. This problem is not trivial. It is particularly difficult when horizontal (time) cuts are made to form the regions, because the mere knowledge that an event may occur at some node is not sufficient. In order to know which processor is to simulate an event, the event's absolute (simulated) time also needs to be known.

Chandy and Sherman propose a solution to this problem. Specifically, they suggest using estimates of the behavior of events in each region, simulating the events in the regions based on the created estimates, sending messages to neighboring regions based on the simulated events to correct the assumptions made, and repeating the simulations to account for the newly arrived messages that correct the original assumptions. This iterative "relaxation" process is repeated until the system reaches equilibrium state at which the messages sent by the processors correspond to the messages that are assumed to be received by processors.

The drawback in the Chandy and Sherman approach is that the initial assumptions made as to the events which occur in a region may be completely wrong. Since the regions are divided a priori in terms of time and nodes rather than by events, three types of errors can be made in the assumptions. Errors that relate to whether events actually occur in the region, errors that relate to the order in which those events occur, and errors that relate to the actual time (vis-a-vis the boundaries of the region) in which the events occur. Because errors in the assumptions will almost certainly be made, and since those errors and the errors they propagate must be corrected, there is little incentive to begin with any assumptions. Indeed, a close analysis of the Chandy and Sherman approach suggests that their iterative "relaxation algorithm" process works no better with some assumptions made than with no assumptions made. The consequence is that processors which handle regions of events far into the future do not perform useful work while the relaxation algorithm either establishes the proper conditions from no assumptions, or establishes the proper conditions by correcting the assumed conditions.

Another drawback of the Candy and Sherman paper is their concentration on the rectangle as a whole and the separation of the rectangle into regions. While they offer some interesting insights of the simulation task as a whole, they provide no suggestions on what are "good" region selections and what processor assignments provide faster and more efficient simulations. Consequently, although speed of simulation is one of the primary goals of simulation methods, a person who performs simulations using the Chandy and Sherman teachings but with some arbitrarily selected regions will not be likely to get the desirable effect of a high speed of simulations.

SUMMARY OF THE INVENTION

Efficient simulation is achieved, in accordance with the principles of this invention, by employing a highly efficient ordering of the events to be simulated. Specifically, the events to be simulated are grouped into layers and the layers are simulated in order. Each of the layers consists of events that are either strictly independent of the other events in the layer or are dependent on other events in the layer but possess a particular attribute. That attribute is a particular timing relationship which relates the times of the events to one another using an associative operator. In complex situations, additional

dependencies between these times may be accommodated by iterative application of procedures involving merging or sorting of the events. Use of the associative operator allows the simulation of N events in a smaller number of computation iterations, such as in O(log N) computation iterations. An operator is associative when the same result is reached whether the operator is applied to a first intermediate result and event C, or to event A and a second intermediate result—where the first intermediate result is obtained by applying the operator to events A and B, and the second intermediate result is obtained by applying the operator to events B and C. One approach for creating simulation layers is to separate the events by the simulated nodes (rather than by time segments, as is typically done in prior art simulations).

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 illustrates a simple four-node serial connection of workstations;

FIG. 2 depicts the simulated job arrival and departure times for the FIG. 1 system;

FIG. 3 presents one structural arrangement for implementing the simulations of FIG. 2;

FIG. 4 illustrates a four-node serial connection of workstations with a merging input;

FIG. 5 illustrates a simple four-node serial connection of workstations with feedback;

FIG. 6 depicts the simulated job arrival and departure times for the FIG. 5 system during the second iteration;

FIG. 7 depicts the final simulated job arrival and departure times for the FIG. 5 system; and

FIG. 8 presents an example of a somewhat more complex system and how the node connections affect the order of simulations.

DETAILED DESCRIPTION

To better understand our invention it is useful to center the description around a specific example. For illustrative purposes, it is assumed, that we need to simulate a system of four workstations (nodes) that are connected serially and together perform useful tasks. Depending on the application, a workstation may be a computer, a worker in a production line, a robot, etc. In this arrangement, "jobs" are applied to the first station in the serial connection at random times, and the arriving jobs have a certain mean and variance of arrival times. Each station processes the arriving jobs on a first-come first-served basis as it becomes free to do the processing. The time that is required to complete each arriving job is random. This randomness also has a certain mean and variance. When a new job arrives at a node before the previous job ended its processing, the new job is placed in a queue. Once the job is completed, it is forwarded to the next node in the serial connection, and the oldest job in the queue is taken up for processing. This arrangement is represented by FIG. 1 in the form of a directed graph. Nodes 10 through 40 represent the station, and the arrows represent the job paths.

The statistics of the arriving jobs in the FIG. 1 arrangement (e.g. the mean and variance of job arrival time intervals) are known, as well as the statistics of the time required by the FIG. 1 station to process the arriving jobs. The challenge is to efficiently determine, through simulation, the times at which jobs arrive and depart each workstation, and subsequently, to efficiently develop the histories and other statistics of the of the job queues in the workstations. It may be also required to compute various statistics, e.g., the mean queue length.

FIG. 2 depicts a possible scenario of job arrivals, processing and completions in the system of FIG. 1. The scenario of FIG. 2 was created randomly for the illustrative purposes of this disclosure. The horizontal axis in FIG. 2 is devoted to nodes and the vertical axis is devoted to simulated time. This configuration is the same as the one described by Chandy and Sherman in the aforementioned article. Each vertical line represents the time line of one of the nodes. Specifically, line 11 represents the time line of node 10, line 21 represents the time line of node 20, line 31 represents the time line of node 30, and line 41 represents the time line of node 40. In FIG.2, each staircase-like shape represents a job. The arrow and the lower edge of the shape represents the job's arrival, and the upper-most horizontal edge of the shape represents the job's departure. The stacking of the staircase-like shapes provides a measure of the queues at the nodes. For example, the job arrivals at node 21 correspond to the job departures at node 11; and, between the arrival of job 16 and the departure of the job corresponding to the arrival of job 22 at node 21, the queue length at node 11 is three jobs.

Although FIG. 2 shows all of the events in the time-space rectangle bounded by the x and y axes and by dashed lines 12 and 13, it should be understood that when simulation starts, neither the existence nor the times of occurrence of these events are known. This information is the immediate result of the simulation. The ultimate answers pertaining to the developed queues come from analyzing the simulated events of the completed FIG. 2. These ultimate answers are relatively easy to come by once the FIG. 2 events are simulated and, consequently, it is the efficient and fast simulation of events that this invention addresses.

As indicated above, one approach is to simulate early events first because the early events dictate the later events. This is the "natural" order. In accordance with one approach of our invention, however, simulations are performed in time slices that are explicitly defined, and the simulations are carried out without the need for "roll-back". Thus, with reference to FIG. 2, simulations begin by simulating events within time interval 0 to t_1, where t_1 is a preselected simulated time on the time axis of FIG. 2. Setting the simulation "horizon" to t_1 causes the simulation of job arrival events 14, 15, and 16 on line 11, the simulation of job arrival event 22 on line 21, the simulation of job arrival event 32 on line 31, and no events on line 41. Event 14 represents the arrival of job 0 at node 10, event 15 represents the arrival of job 1 at node 10, event 16 represents the arrival of job 2 at node 10, event 22 represents the arrival of job 0 at node 20, and event 32 represents the arrival of job 0 at node 30. The job departure times are also events that occur at nodes 10, 20, 30 and 40. Thereafter, having completed a slice the simulation "horizon" is advanced to a later time, such as time t_2 and events 17, 23, 24, 33, 34, 42, 43, and 44, are simulated. The process continues until time t_9, whereupon the simulating of all of the FIG. 2 events is completed. This horizontal slicing which results from the successive selection of "horizons" forms, in effect, simulation layers that are processed seriatim.

In accordance with another approach of our invention, the events to be simulated are also divided into layers and the layers are simulated seriatim. However, the layers in this approach are selected in a very differ-

ent manner. Specifically, each layer contains the events whose simulations depend on no other events, depend on events that were simulated in previous layers, or depend on events in the current layer that belong to class X. Events are in class X when they can be segregated into groups and ordered so that events in a group can be simulated from events in the previous groups with the aid of an associative operator. The ordering of groups can be in a linear sequence $1\rightarrow 2\rightarrow \ldots \rightarrow i-1\rightarrow \ldots$ as in the simple case considered here, but can also be more complex; e.g., it may constitute a tree. In all cases, however, the notion of a "previous" group must be defined. In the case of a linear sequence, the previous group for group i is group $j<i$. One consequence of this attribute (belonging to class X) is that n events can be simulated in O(log n) computation iterations. When the number of the groups within a layer is large, substantial simulation speed-up is derived from the O(log n) attribute. The number of events in the conventional time slice layers is very small, and its parallelism is bounded. Not much benefit is derived, therefore, from the O(log n) attribute, even if the events do belong to class X. Thus, the benefits of our first-mentioned approach derive primarily from a different aspect of our invention. On the other hand, the number of events in a time slice along the time line is unbounded, and hence its parallelism is unbounded. Events along a time line do belong to class X, as demonstrated below.

With reference to FIG. 2, the following will show how to compute the sequence of departure events for each of the nodes in the system of FIG. 2. Computing the corresponding sequence of arrival events given the inter-arrival periods is similar. Merging the arrival and departure sequences and summing over the merged sequence treating arrivals as +1's and departures as −1's, procedures that can be done with great efficiency on a multiprocessor, determines the queue length history shown in the figure. The time at which the i^{th} job departs from a node (and arrives at the next node) can be expressed as

$$D_i=\max(A_i, D_{i-1})+S_i \tag{1}$$

where A_i is the time of arrival of job i at the given node, and S_i is the service interval for job i. Equation (1) can also be written as

$$D_i=\max((A_i+S_i),(D_{i-1}+S_i)), \tag{2}$$

and equation (2) can be written as

$$D_i=A_i S_i+D_{i-1}\cdot S_i \tag{3}$$

where unlike normal notations the "·", or product, operation represents addition, and the "+" operation represents the "max" function. Proceeding from here,

$$\begin{aligned} D_{i+k}=&A_{i+k}S_{i+k}+A_{i+k-1}S_{i+k}S_{i+k-1}+\ldots \\ &A_{i+1}S_{i+k}S_{i+k-1}S_{i+k-2}\cdots \\ &S_{i+1}+D_iS_{i+k}S_{i+k-1}S_{i+k-2}\cdots S_{i+1} \end{aligned} \tag{4}$$

Written in closed form, the above becomes

$$D_{i+k}=\sum_{j=i+1}^{i+k} A_j \prod_{r=j}^{i+k} S_r + D_i \prod_{r=i+1}^{i+k} S_r \tag{5}$$

Since equation (5) expresses D_{i+k} in terms of D_i, a clearer way to express equation (5) may be through a function H_i^{i+k} which operates on D_i to derive the value of D_{i+k}; i.e.,

$$D_{i+k}=H_i{}^{i+k}(D_i). \tag{6}$$

The question is whether the H operator (operating on the variables identified by the subscript and the superscript) can be expressed in term of the composition of H operators. If so, that would imply that $H_{i1}{}^{i4}(D)$ can be determined by evaluating $H_{i2}{}^{i4}(H_{i1}{}^{i2}(D))$ or by evaluating $H_{i3}{}^{i4}(H_{i1}{}^{i3}(D))$. To demonstrate that this is true, we note that $H_{i3}{}^{i4}(H_{i1}{}^{i3}(D))$ equals

$$\sum_{j=i3+1}^{i4} A_j \prod_{r=j}^{i4} S_r + \prod_{r=i3+1}^{i4} S_r\left(\sum_{j=i1+1}^{i3} A_j \prod_{r=j}^{i3} s_r + \prod_{r=i1+1}^{i3} S_r(D_i)\right). \tag{7}$$

Combining terms, we get, as expected,

$$\sum_{j=i1+1}^{i4} A_j \prod_{r=j}^{i4} S_r + D_{i1} \prod_{r=i1+1}^{i4} S_r \tag{8}$$

Similarly, $H_{i1}{}^{i2}(H_{i1}{}^{i2}(D))$ equals

$$\sum_{j=i2+1}^{i4} A_j \prod_{r=j}^{i4} S_r + \prod_{r=i2+1}^{i4} S_r\left(\sum_{j=i1+1}^{i2} A_j \prod_{r=j}^{i2} s_r + \prod_{r=i1+1}^{i2} S_r(D_i)\right) \tag{9}$$

which also equals

$$\sum_{j=i1+1}^{i4} A_j \prod_{r=j}^{i4} S_r + D_{i1} \prod_{r=i1+1}^{i4} S_r \tag{10}$$

Thus, equation (3) becomes $D^i=H_{i-1}{}^i(D_{i-1})$ and equation (6) can be interpreted as

$$\begin{aligned} H_i{}^{i+k}=&H_{i+k-1}{}^{i+k}\cdot H_{i+k-2}{}^{i+k-1}\cdot \ldots \\ &H_{i+1}{}^{i+2}\cdot H_i{}^{i+1}, \end{aligned}$$

where · denotes function composition. Since function composition is associative, we can group the functions in any advantageous manner.

In light of the above, layers that correspond to the time lines of FIG. 2 constitute one valid layering approach in accordance with the principles of this invention. When such layers are selected, the departure times of the jobs along a time line can be evaluated iteratively as shown in the table below.

job	initial		iteration 1		iteration 2		iteration 3	
i	T(i)	π(i)	T(i)	π(i)	T(i)	π(i)	T(i)	π(i)
7	H_6^7	6	$H_6^7H_5^6 = H_5^7$	5	$H_5^7H_3^5 = H_3^7$	3	$H_3^7H_{-1}^3 = H_{-1}^7$	−1
6	H_5^6	5	$H_5^6H_4^5 = H_4^6$	4	$H_4^6H_2^4 = H_2^6$	2	$H_2^6H_{-1}^2 = H_{-1}^6$	−1
5	H_4^5	4	$H_4^5H_3^4 = H_3^5$	3	$H_3^5H_1^3 = H_1^5$	1	$H_1^5H_{-1}^1 = H_{-1}^5$	−1
4	H_3^4	3	$H_3^4H_2^3 = H_2^4$	2	$H_2^4H_0^2 = H_0^4$	0	$H_0^4H_{-1}^0 = H_{-1}^4$	−1
3	H_2^3	2	$H_2^3H_1^2 = H_1^3$	1	$H_1^3H_{-1}^1 = H_{-1}^3$	−1	H_{-1}^3	−1
2	H_1^2	1	$H_1^2H_0^1 = H_0^2$	0	$H_0^2H_{-1}^0 = H_{-1}^2$	−1	H_{-1}^2	−1
1	H_0^1	0	$H_0^1H_{-1}^0 = H_{-1}^1$	−1	H_{-1}^1	−1	H_{-1}^1	−1
0	H_{-1}^0	−1	H_{-1}^0	−1	H_{-1}^0	−1	H_{-1}^0	−1

The table demonstrates that, when there are 8 jobs, the necessary number of iterations required for developing all of the job departure times, when grouped in an advantageous manner, is bounded by $\log_2 8$, or 3. At each iteration, operators H which determine departure times are computed for jobs in the queue based on a previous computation. Each iteration also provides a pointer π(i) for computing the operators at the next iteration. Thus, in connection with job 6, for example, the initial value is H_5^6 and the pointer $\pi(6)$ is set to 5. At the first iteration, the computation is based on the value in the row of event 5, operator H_4^6 is computed and the pointer $\pi(6)$ is set to the current pointer value $\pi(5)$ of event 5; to wit, to 4. At the second iteration, operator H_2^6 is computed based on the value in the row of event 4 and the pointer $\pi(6)$ is set to the current pointer value $\pi(4)$ of event 4; that is, 2. Finally, in the third iteration, operator H_{-1}^6 is computed based on the values in the row of event 2, and the pointer $\pi(6)$ is set to the current pointer value $\pi(2)$ of event 2, i.e., −1. Having the complete set of operators H_{-1}^0, H_{-1}^1, . . . H_{-1}^7 at iteration 3, we can immediately compute the set of departure times $D_0 = H_{-1}^0 D_{-1}$, . . . $D_7 = H_{-1}^7 D_{-1}$, where D_{-1} is assumed 0.

The above table demonstrates that the events of a node can be advantageously assigned to one layer. Typically, a simulation effort involves simulating a large number of events in a node, and the assigning of a large number of events to one layer could appear, on first blush, to be disadvantageous. Actually, this works to the advantage of this invention because additional processing power can always be applied to simulate the node events in parallel, and because the class X attribute permits the overall processing of a layer to grow only as the logarithm of the number of simulated events. In an ultimately parallel environment, a processor can be devoted to the simulation of each single event in the layer (as compared to the prior art devoting of a processor to the simulation of a single node). Once a layer has been simulated, information is available to simulate the next layer. In connection with the FIG. 2 events and a layering approach based on vertical strips, the first layer would correspond to line 11, the second layer would correspond to line 21, and so forth.

In connection with the FIG. 2 events and a layering approach based on horizontal strips, the first layer would correspond to the strip bounded by (0, t_1), the second layer would correspond to the strip bounded by (t_1, t_2), and so forth.

A system for performing the simulations in accordance with the principles of our invention can have various architectures. It can comprise a single computer or many computers that are connected in a network that permits communication between the computers. FIG. 3 presents a simple organization of computers 100 through 107 that are connected to network 110. Connection network 110 can be a conventional packet transmission network, a space division switch, or the like.

Carrying out the simulations for FIG. 2, the events of time line 11 are all simulated in the system of FIG. 3 and the results are stored in the computers as inputs to the simulation of the second layer—the layer of events along time line 21. The process repeats until the events of time line 41 are simulated. It may be noted in passing that, clearly, it is possible for the number of events in a layer to far exceed the number of processors. That does not present a problem, however, because more than one event can be assigned to each processor. The assignment can be random but, most productively, a number of adjacent events can be assigned to each processor. Moreover, we need not imagine that each event is assigned to a particular processor, but that the intermediate results needed at each iteration are performed cooperatively by the available processors.

It may also be mentioned in passing that the user may not know at what simulated time to terminate the simulation. The decision to end simulations may depend, in fact, on observed results. This situation can be accommodated by selecting large number of events in a slice, treating each slice as a simulation task, and simulating each task in accordance with the disclosed principles. This is an example where the first approach of our invention (explicitly defined horizontal slices) can be combined with the second approach of our invention (vertical slices where the events are in class X). Of course, even when not combined with the second approach, when employing the first approach of our invention the total number of processors are assigned to the simulation of events in the slice. Thus, in connection with FIG. 3 and the events of the first time slice in FIG. 2, the eight processors are assigned to the 5 events in the four nodes. The amount of memory available on the machine used to perform the simulation may constrain the number of events that can be simulated concurrently. In that case, memory can be reused (e.g., using circular buffering) as the simulation proceeds.

The FIG. 1 system is rather simple in that it contains no inputs other than the single input path to node 10. A more complex situation is depicted in FIG. 4, where another input path is provided to node 40. This additional input path, designated by arrow 48, needs to be merged with the input path emanating from node 30.

When the layer that encompasses node 40 is simulated, the structure of FIG. 4 calls for a merging of the events coming to node 40 from the two paths (the path of arrow 48 and the path of arrow 49). Merging of these events is not difficult because their times of arrival are known when merging is to proceed, and each of the input paths supplies a time-ordered, or sorted, list of events. Merging of two sorted lists is well known in the art as described, for example, in Batcher, "Sorting Networks and Their Applications", AFIPS SJCC 32, 1968.

A more complex situation results when there is feedback. Feedback occurs when the output of a node is reapplied to a previous node. Of course, there must be some control in the system to preclude overload and, typically, that involves some test that is performed on the departing jobs to determine whether they should be fed back or not. This test may be based on the nature of the job's output or it may be based on a simple job independent rule, such as "send each job back for repeated processing, but only once".

FIG. 5 depicts a structure that is modeled after the FIG. 1 organization, but with a "once only" feedback path from node 40 to node 10. More specifically, in the FIG. 5 structure each job that receives service at node 10 (line 11 in FIG. 2) terminates or departs from the system. It does not re-enter node 20.

The simulations task of the FIG. 5 arrangement is not as straight forward as that of FIG. 1. The problem is that the departure times of events in node 10 are dependent on the arrival times of events from node 40, and those times are not known and cannot be directly simulated with an associative operator. To resolve this dilemma, we use a relaxation algorithm that is similar to that of Chandy and Sherman.

We start with the assumption that there is no feedback from node 40 and that the only input to node 10 is from the path of arrow 50. We then compute event departure times at node 40 (and the intermediate nodes) based on those assumptions. Thereafter, we merge the developed departure times of node 40 with the arrow 50 input and recompute the nodes' departure times. This iterative process is repeated until the developed events at an iteration are the same as the developed events at the previous iteration.

FIG. 2 and FIGS. 6 and 7 demonstrate the simulation of events for the FIG. 5 system. FIG. 2 represents the first iteration, when no feedback is assumed to be present. FIG. 6 presents the event simulations of the second iteration, and FIG. 7 presents the event simulations of the third and final iteration.

The system depicted in FIG. 1 has one very simple aspect about it, and that is that node 10 is clearly the node where the simulations should start. No other node can receive jobs (directly or indirectly) from a source other than node 10. That is not to say that the simulations task cannot be performed by starting at another node. However, starting at another node would make the task take somewhat longer to complete. For example, one could assume some job entries to node 20, then simulate nodes 30, 40 and 10, iteratively correct the assumptions regarding the inputs to node 20 and the resulting outputs at node 20 until the system reaches the equilibrium.

The system of FIG. 4 has two nodes that accept jobs from outside the system (nodes 10 and 40) but here it is not quite clear that node 10 is the preferred starting node. One could start with node 40 and the inputs arriving at the arrow 49 path, while assuming some inputs from the path of arrow 48, proceeding on the basis of the known and assumed information, and correcting the assumptions when more information is known. The fact that assumptions are made and that an iterative "loop" must be engaged to correct for errors in the assumptions makes it clear that a starting node should, if possible, be one that does not require any assumptions. Stated differently, it should comprise events that are least dependent of all other nodes. In FIG. 4, the "natural" order 10, 20, 30, 40 requires no such assumptions.

The system of FIG. 5 includes a feedback path from node 40 to node 10, and that creates a situation where there are no nodes whose events are independent from all other nodes. Still, node 10 is different from nodes 20–30 in that node 10 includes an independent input from the path of arrow 50. Utilizing the available information as early as possible reduces the need for later corrections and, therefore (all things being equal), it still makes sense to start the simulations with node 10. Of course, when node 40 also had an independent input, as in FIG. 5, then node 10 and node 40 would be topographically equivalent and the simulation could start with either of the two nodes.

From the above it appears clear that the initial task in an event simulation undertaking is to analyze the directed graph that represents the system under consideration, and the rate of the flows expected along those paths. The analysis should develop an ordered list of the events to be simulated in such a way that the simulation layers encompass the largest possible number of events. When chosing to create layers that correspond to the events of nodes (as described above), the analysis should develop an ordered list of the nodes to be simulated. When developing an ordered list of nodes, the top of the list should include the nodes that are least dependent of all other nodes. This should be followed by the nodes that depend thereon, in the order of the nodes having the fewest number of input paths that depend on nodes that are below that node (in the ordered list).

FIG. 8 presents an example. Therein, nodes 74 and 76 are found to have only independent inputs and, accordingly, they are selected first. Their relative order is unimportant. Nodes 75 and 73 are found to be dependent on nodes 74 and 76 and, therefore, they should be selected next. However, node 73 is also dependent on nodes 72 and 71, which as yet are not in the list and hence potentially below node 73 in the list, while node 75 is dependent on no nodes that are not already included in the list. Accordingly, node 75 is selected first, followed by node 73. At this point the list comprises, in order, nodes 74, 76, 75, and 73. Next, node 71 is determined to be the only remaining node that depends on any of the nodes included in the list (depends on node 73) and, therefore, it is selected next for the list. Lastly, node 72 is selected because it depends on node 71 (and because it is the last remaining node to be included in the list). The above procedure is circumscribed by the following two procedures:

1. select the node(s) that depend on no unknown inputs or, stated differently, that depend on no nodes that are not already in the list. Repeat until all nodes are exhausted or until no some nodes remain but the remaining nodes fail to meet the above criteria. In such a case, go to step 2.
2. Of the remaining nodes, from among the nodes that depend on nodes that are already in the list, select the node that depends on the fewest number of nodes that are not already in the list. If there is more than one such node, select one arbitrarily. Having made the selection, return to step 1.

Having created the basic list, the procedure is to execute the simulations in the list, and to repeat the simulations in the list, in order, until the equilibrium is reached. We found that the needed number of repetitions is very small; usually, on the order of log N, where N is the number of events to be simulated.

Generalizing on the above, any directed graph can be analyzed to form an ordered list as described above.

Creating the list when the graph is acyclic is straight forward. When the graph has cycles, however, the situation is somewhat different.

One approach for dealing with a directed graph that include cycles is to redraw the graph in a form that is devoid of cycles. This is accomplished by representing the cycles encompassed by each strongly connected component as a single node. A strongly connected component is the set of all nodes, and the corresponding links, that are mutually reachable from any of the nodes within the set. Having developed an acyclic representation of the directed given, parent, graph, the ordered list can be created and simulated. In the course of simulating the strongly connected components in the list, the structure of the strongly connected component is analyzed as an independent graph with the appropriate entry points from the parent graph. The event list developed for the strongly connected component is simulated the necessary number of times until equilibrium is reached before the next node in the parent list is simulated.

The above description presents the principles of this invention by way of specific techniques and examples, but it should be appreciated that many variations can be made without departing from the spirit and scope of this invention. For example, when chosing to develop simulation layers by creating vertical strips that encompass the events of a node, there is no reason to insist that each strip include one and only one node. Indeed, in connection with FIG. 8, the analysis revealed the interesting situation that nodes 76 and 74 are interchangeable in the ordered list. That implies that nodes 74 and 76 can be combined into a single layer. When enough processing power is available, such combining can further speed up the simulations.

Another interesting situation results when the entire directed graph is a strongly connected component without any entry points. An example of that may be a ring network of 5 workstations with 10 tokens constantly circulating from one workstation to the next. It can be shown that the 50 events which represent the set of 10 token-processing jobs that pass through the 5 workstations can be considered as a group, because an associative operator can be found for the resulting groups. This 50-event group is qualitatively similar to the one-event group represented above by D_i. The number of such groups will be of order of N where N is the number of simulated events, so the available parallelism, as before, is still of order of N.

Still another interesting situation results when workstations have limited input buffers which prevent the nodes from creating boundless job queues. It can be shown that in this situation, order of N groups of certain sets of events may be formed across all nodes, and that an associative operator can be found for these groups.

Yet another interesting situation that benefits from the principles of this invention is found in the ALOHA protocol. In the ALOHA arrangement, a plurality of workstations communicate over a common channel. When a collision of communication packets occurs, the protocol calls for the colliding workstations to be informed of the collision. Each of the involved stations waits a random interval and then attempts to retransmit. This situation employs the associative operator to determine the simulated departure times, and merging procedure to reinsert the retransmission events into the events set. The number of groups for application of the associative operator is of order N, so it would take order of log N iterations to complete the simulation.

Event situations where different jobs have different priorities can be handled with this invention. For example, when jobs appear with one of two priorities, the higher priority jobs can be handled first, as if the other priority jobs did not exist, and then the lower priority jobs would be handled.

Another aspect of job simulations that imparts a "priority" flavor is related to the physical limitations that the simulation system of FIG. 3 may possess. Specifically, all systems in today's architectures have a memory size limitation, and this memory limitation may impose an ordering of the simulated events. For example, in connection with the FIG. 4 arrangement, if the events arriving at node 10 arrive earlier than the events arriving at node 40 (path 49) and the number of such events is large enough to pose a memory overflow concern, it makes sense to devote more resources to simulating the events of node 10 than to simulating the events at node 40. Because of the complexity inherent in the novel concept of the "associative operator", the above description concentrates on the "vertical" layers are applicable to other layering approaches including, specifically, the "horizontal" layering approach described above. Indeed, it may be pointed out that in some situations the second approach of our invention, where "horizontal", or "time", slices are selected in an explicitly defined manner and all of the processing power is devoted to simulating the slices to completion in a seriatim manner, may be better than the "vertical" layering approach. As an example of such a situation, one might have (some time in the future) a massively parallel processor with perhaps 1,000,000 individual processors working in parallel, for the system to be simulated consists of 1,000 nodes. Assigning all 1,000,000 computing units for processing events in the first node, then the second node, in the chosen order is clearly possible (in accordance with the above-described principles of our second approach) it may be not advantageous. There simply may not be a need to simulate the system a million events into the future of each node. Instead, one might employ the first approach of our invention, define horizontal time slices (perhaps so that each node has about 1,000 events to process in each slice) and assign the 1,000,000 processors to simulate the 1,000 events in each of the 1,000 nodes.

We claims:

1. A method for simulating on a computer events of a system comprising the steps of:

selecting a cluster of events of said system that includes primarily events that are related to each other through an associative operator, an operator being associative when the same result is reached whether the operator is applied to a first intermediate result and event C, or to event A and a second intermediate result—where the first intermediate result is obtained by applying the operator to events A and B, and the second intermediate result is obtained by applying the operator to events B and C;

simulating the events of said cluster of events; and

returning to said step of selecting when at least some of said events of said system have not been simulated.

2. The method of claim 1 wherein said step of selecting excludes events that belong to another cluster.

3. The method of claim 1 wherein said step of simulating, when simulation information about events in other clusters is required, utilizes the simulation information of said events in other clusters developed by preceding steps of simulating, and makes assumptions about the simulation information of said events in other clusters to which said step of simulating was not applied.

4. The method of claim 3 wherein said steps of selecting, simulating and returning form a sequence of cluster simulations, and at least a portion of said sequence is repeated until the simulation information for all of the events in repetition k of said sequence, where k is an integer, is the same as the simulation information for all of the events in repetition k-1 of said sequence.

5. A method for simulating on a computer events of a system comprising the steps of:

selecting a layer of events of said system that includes mostly event groups that are related to each other through an associative operator, an operator being associative when the same result is reached whether the operator is applied to a first intermediate result and event C, or to event A and a second intermediate result—where the first intermediate result is obtained by applying the operator to events A and B, and the second intermediate result is obtained by applying the operator to events B and C;

simulating said layer of events; and

returning to said step of selecting when at least some of said system events have not been simulated.

6. The method of claim 5 wherein said step of selecting excludes events that belong to another layer.

7. The method of claim 5 wherein said step of simulating, when simulation information about events in other clusters is required, utilizes the simulation information of said events in other clusters developed by preceding steps of simulating, and makes assumptions about the simulation information of said events in other clusters to which said step of simulating was not applied.

8. The method of claim 5 wherein said steps of selecting, simulating and returning form a sequence of cluster simulations, and at least a portion of said sequence is repeated until the simulation information for all of the events in repetition k of said sequence, where k is an integer, is the same as the simulation information for all of the events in repetition k-1 of said sequence.

9. The method of claim 5 wherein the event groups that are related to each other through an associative operator are such that the groups can be ordered and the events of a group can be simulated from the events of some previous groups with the aid of said associative operator.

10. The method of claim 5 wherein said system comprises a plurality of interacting nodes and each of said layers comprises primarily the events of one of said nodes.

11. The method of claim 5 wherein said system comprises a plurality of interacting nodes and each of said layers consists the events of one of said nodes.

12. The method of claim 5 wherein said step of simulating a layer includes a step of merging the events of other layers that affect said layer.

13. The method of claim 5 wherein said step of simulating a layer includes the steps of

merging the simulated events of layers that affect said layer and that have been previously simulated; and

merging assumed events of layers that affect said layer but which have not been previously simulated.

14. The method of claim 13 wherein said steps of selecting, simulating and returning from a sequence of cluster simulations, and at least a portion of said sequence is repeated until the simulation information for all of the events in repetition k of said sequence, where k is an integer, is the same as the simulation information for all of the events in repetition k-1 of said sequence.

15. A method for discrete event simulation on a computer of system events occurring in a plurality of nodes in a multi-node system, where events in one node correspond to a time interval having more than one time sample and affect events in another node, comprising the steps of:

selecting the events of a node;

simulating events of the selected node; and

returning to said step of selecting until the last node in said order has been simulated and all events have been simulated.

16. The method of claim 15 wherein said simulating comprises simulating departure times of jobs.

17. The method of claim 15 wherein said simulating is carried out with the aid of an associative operator.

18. The method of claim 17 wherein said associative operator includes a "max" function which selects the larger of said two inputs.

19. The method of claim 15 wherein said step of selecting includes a step of formulating a simulations order of said nodes.

20. The method of claim 19 wherein said step of formulating a simulations order is based on an acyclic directed graph representation of said system, which graph comprises at least one node from the set of nodes that includes strongly connected component nodes and non-strongly connected component nodes.

21. A method for discrete event simulation on a computer of system events occurring in a plurality of nodes in a multi-node system, where events in one node correspond to a time interval having more than one time sample and affect events in another node, comprising the steps of:

formulating a simulations order for simulating said nodes;

designating said the first node in said order as the simulation node;

simulating events scheduled for said simulation node;

designating a new simulation node by selecting the node that follows, in said order, the current simulation node; and

returning to said step of simulating until the last node in said order has been simulated and all events have been simulated.

22. The method of claim 21 wherein said step of formulating a simulations order comprises:

attaining a directed parent graph to represent the interaction of said plurality of nodes;

creating an acyclic directed graph from said directed parent graph by representing each strongly connected component in said parent graph by node;

selecting a node in said acyclic directed graph that is dependent on the fewest number of other nodes in said acyclic directed graph;

assigning the selected node to a list when the selected node is nonstrongly connected component node;

developing a sublist when the selected node is a strongly connected component node and appending said sublist to said list; and

returning to said step of selecting until the last node in said acyclic directed graph has been selected.

23. A method for discrete event simulation on a computer of system events occurring in a plurality of nodes in a multi-node system, where events in one node correspond to a time interval having more than one time sample and affect events in another node and where the termination of the appearance on independent events at one or more of the system nodes is not known, comprising the steps of:

selecting a super-group of events;

for events within the selected super-group, executing a simulation procedure including

selecting a cluster of events from among the events within the selected super-group, which cluster includes primarily events that are related to each other through an associative operator;

simulating the events of said cluster of events; and returning to said step of selecting when at least some of said events of said system have not been simulated;

selecting another super-group of events; and

returning to said step of executing a simulation procedure.

24. The method of claim 5 wherein said step of simulating develops an order of simulating the events in said layer.

25. The method of claim 24 wherein said order is related to priority of the events being simulated.

26. The method of claim 24 wherein said order is related to constraints of hardware in which said method of simulating is carried out.

27. The method of claim 24 wherein said order is related to memory constraints of hardware in which said method of simulating is carried out.

28. The method of claim 1 wherein said step of selecting selected the events of a cluster based on said associative operator.

29. The method of claim 1 wherein said step of simulating simulates the events with the aid of said associative operator.

30. The method of claim 1 wherein said step of selecting selected the events of a cluster based on said associative operator, and said step of simulating simulates the events with the aid of said associative operator.

31. A method for simulating events of a system with an available number of processors comprising the steps of:

dividing the events to be simulated into layers having defined interface borders between the layers where at least in some of the layers one of the events in a layer is causally related to at least one other event in the layer;

selecting an order of simulation for simulating said layers; and

simulating the layers in a seriatim manner, where each step of simulating a layer completely simulates the layer by employing essentially all of the available number of processors.

32. The method of claim 31 wherein said step of dividing the events to be simulated creates clusters of events where each cluster of events of said system includes primarily events that are related to each other through an associative operator.

33. The method of claim 31 wherein said step of dividing the events to be simulated creates clusters of events where each cluster of events of said system includes events within a lower border and an upper border of the simulated time where the lower border and the upper border encompass events of more than one particular time.

* * * * *

United States Patent [19]

Lubachevsky

[11] **Patent Number: 4,901,260**

[45] **Date of Patent: Feb. 13, 1990**

[54] **BOUNDED LAG DISTRIBUTED DISCRETE EVENT SIMULATION METHOD AND APPARATUS**

[75] Inventor: **Boris D. Lubachevsky**, Bridgewater, N.J.

[73] Assignee: **American Telephone and Telegraph Company AT&T Bell Laboratories**, Murray Hill, N.J.

[21] Appl. No.: **114,369**

[22] Filed: **Oct. 28, 1987**

[51] **Int. Cl.**[4] **G06F 15/16**

[52] **U.S. Cl.** **364/578**; 371/23

[58] **Field of Search** ..... 364/578; 340/286 M, 340/515; 371/23

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,204,633	5/1980	Goel	371/23
4,527,249	7/1985	Van Brunt	371/23
4,636,967	1/1987	Bhatt et al.	364/552
4,644,487	2/1987	Smith	364/578
4,680,784	7/1987	Lehnert et al.	379/11
4,751,637	6/1988	Catlin	364/200

OTHER PUBLICATIONS

Abramovici et al., "A Logic Simulation Machine", IEEE Transactions on CAD of IC's & Systems, vol. 2 No. 2 Apr. '83, pp. 82–93.

Howard et al. "Parallel Processing interactively Simulates Complex VSLI Logic", Electronics Dec. 15, 1983, pp. 147–150.

IEEE Transactions on Software Engineering, vol. SE–5, No. 5, Sep. 1979, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", K. M. Chandy and J. Misra, pp. 440–452.

Computer Networks, vol. 3, No. 1, Feb. 1979, "Distributed Simulation of Networks", K. M. Chandy, V. Holmes, and J. Misra, pp. 105–113.

Communications of the ACM, vol. 24, No. 11, Apr. 1981, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", K. M. Chandy and J. Misra, pp. 198–206.

Proceedings of the Society for Computer Simulation, (SCS), Distributed Simulation Conference, Jan. 1985, "Fast Concurrent Simulation Using the Time Warp Mechanism", D. Jefferson and H. Sowizral, pp. 63–69.

Proceedings of the Int. Conf. on Modelling Techniques and Tools for Performance Analysis, May 16–18, 1984, Paris, "Parallel Time-Driven Simulation of a Network on a Shared Memory MIMD Computer", B. D. Lubachevsky and K. G. Ramakrishnan.

Primary Examiner—Parshotam S. Lall
Assistant Examiner—Christopher L. Makay
Attorney, Agent, or Firm—Henry T. Brendzel

[57] **ABSTRACT**

A discrete event simulation system that avoids all blocking and advances the simulation time in an efficient manner by treating the simulated system as a set of subsystems and simulating the subsystems concurrently. The simulation proceeds iteratively by restricting the simulation of scheduled events for each subsystem at any one time to a chosen simulated time segment (bounded lag) beginning with the lowest simulation time found among the subsystems. With each simulation iteration, an "at risk", demarcation time is evaluated based only on a subset of the subsystems that can potentially affect the simulation at the considered subsystem. Events scheduled for a time earlier than the "at risk" time are simulated. In simulating systems where some subsystems affect other subsystems only through intermediate subsystems, opaque periods can be experienced when, because of the specific process that is being simulated, such an intermediate subsystem "promises" that a particular route emanating from this subsystem would be busy for a set period of time, and thereby also "promises" that no other subsystem can use this route as a conduit to affect other subsystems. That tends to push forward the "at risk" demarcation time.

24 Claims, 3 Drawing Sheets

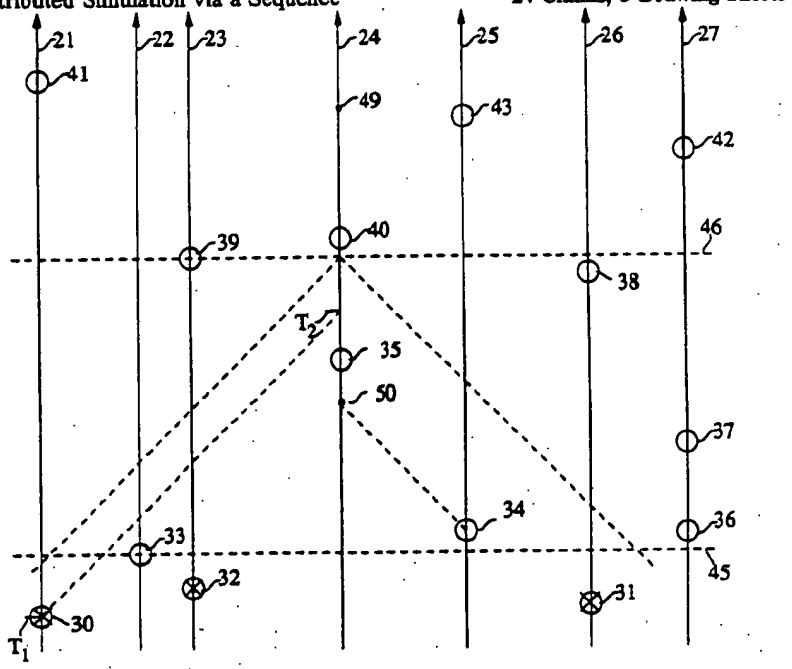


FIG. 1

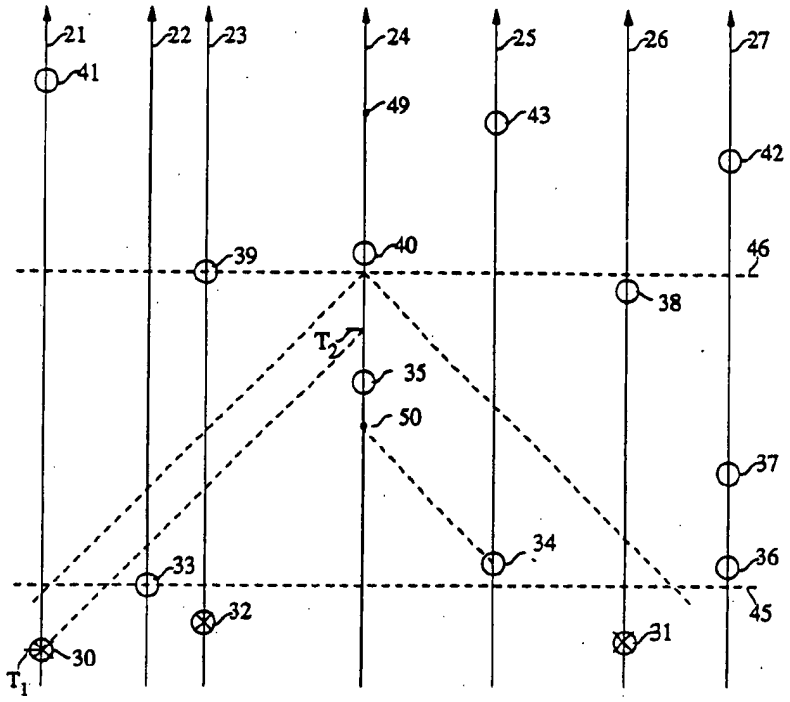

21
22
23
24
25
26
27
41
49
43
42
39
40
46
38
T_2
35
50
37
34
36
33
32
45
31
30
T_1

FIG. 2

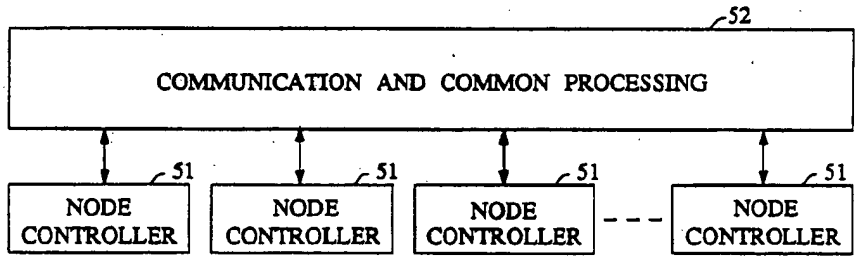

52
COMMUNICATION AND COMMON PROCESSING
51
NODE CONTROLLER
51
NODE CONTROLLER
51
NODE CONTROLLER
51
NODE CONTROLLER

FIG. 3

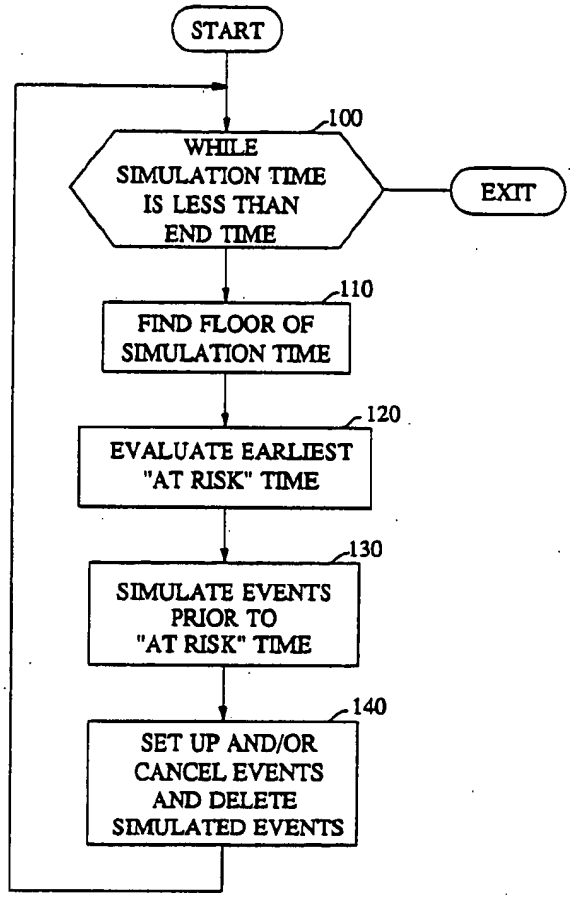
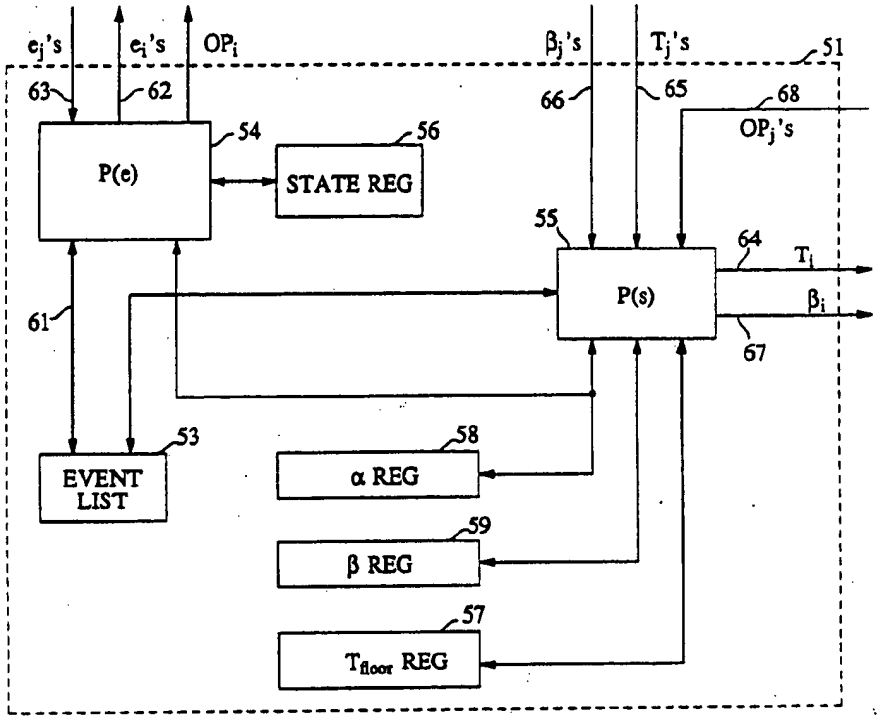

START
100
WHILE SIMULATION TIME IS LESS THAN END TIME
EXIT
110
FIND FLOOR OF SIMULATION TIME
120
EVALUATE EARLIEST "AT RISK" TIME
130
SIMULATE EVENTS PRIOR TO "AT RISK" TIME
140
SET UP AND/OR CANCEL EVENTS AND DELETE SIMULATED EVENTS

FIG. 4


e_j's
e_i's
OP_i
β_j's
T_j's
51
63
62
66
65
68
OP_j's
P(e)
54
56
STATE REG
55
64
T_i
P(s)
β_i
61
67
53
58
EVENT LIST
α REG
59
β REG
57
T_{floor} REG

BOUNDED LAG DISTRIBUTED DISCRETE EVENT SIMULATION METHOD AND APPARATUS

BACKGROUND OF THE INVENTION

This invention relates to the simulation art, and more particularly to the art of distributed discrete event simulation.

Computer simulation has become very important in recent years because of the many applications where simulation of systems is highly beneficial. One such applicaton is the use of simulation in the design of complex systems. These may be electronic systems such as a telecommunicatons switching network, robot based flexible manufacturing systems, process control systems, health care delivery systems, transportation systems, and the like. Design verification through simulation plays an improtant role in speeding up the design and insuring that it conforms to the specification. Another application is the use of simulation in analyzing, and tracking down, faults appearing in operating system. Still another application is optimizing the operation of existing systems through repeated simulations, e.g., the operation of a manufacturing facility, the operation of the telecommunications network, scheduling and dispatching, etc. Yet another application is the use of simulation to predict the operation of systems which for various reasons can not be tested (e.g., resonse to catastrophe).

Simulations can be classified into three types: continuous time, discrete time, and discrete event. Discrete event simulation means simulation of a system in which phenomena of interest change value or state at discrete moments of time, and no changes occur except in response to an applied stimulus. For example, a bus traveling a prescribed route defines a discrete event system in which the number of passengers can change only when the bus arrives at a bus stop along the route.

Of the three simulation classes, from computation standpoint it appears that discrete event simulation is potentially the least burdensome approach because simulation of time when nothing happens is dispensed with. Of course, synchronizaton of the event simulations must be considered when parellelism is employed. Most often, a discrete event simulator progresses by operating on an event list. An event at the top of the list is processed, possibly adding events to the list in the course of processing, and the simulation time is advanced. Thereafter, the processed event at the top of the list is removed. This technique limits the speed of simulation to the rate at which a single processor is able to consider the events one at a time. In a parrellel scheme many processors simultaneously are engaged in the task creating a potential for speeding up the simulation. Although techniques for performing event list manipulation and event simulation in parallel have been suggested, large scale performance improvements are achieved only by eliminating the event list in its traditional form. This is accomplished by distributed simulations.

In a distributed simulation, a number of parallel processors form a simulation multicomputer network, and it is the entire network that is devoted to a simulation task. More specifically, each processor within the network is devoted to a specific portion of the system that is simulated; it maintains its own event list and communicates event occurrences to appropriate neighbor processors. Stated conversely, if one views a simulated system as a network of interacting subsystems, distributed simulation maps each subsystem onto a processor of the multicomputer network.

Although distributed simulation provides parallelism which has the potential for improving the simulation speed, allocation and synchronization of work among the processors is a major concern which may impede the realization of the improvement goals. One well known approach for distributed simulation has been proposed by Chandy and Misra in "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," IEEE Transactions on Software Engineering, Vol. SE-5, No. 5, September 1979, pp. 440–452, and by Chandy, Holmes and Misra in "Distributed Simulation of Networks," Computer Networks, Vol. 3, No. 1, February 1979 pp. 105–113. In this approach, they recognize that physical systems to be simulated are composed of independent but interacting entities, and that those entities should be mapped onto a topologically equivalent system of logical nodes. Interaction between nodes is accomplished by the exchange of time-stamped messages which include the desired message information and identify the simulation time of the sending node. In accordancewith the Chandy-Holmes-Misra approach, the nodes interact only via messages. There are no global shared variables, each node is activated only in response to a message, each node maintains its own clock, and finally, the time-stamps of the messages generated by each node are non-decreasing (in time). In this arrangement, each of the nodes works independently to process the events assigned to it in the correct simulated order. Thus, independent event can be simulated in parallel, within different nodes, even if they occur at different simulated times.

The time stamping is required, of course, to maintain causality so that in a message-receiving node an event that is scheduled for time T is not simulated when other incoming messages can still arrive with a time-stamp of less than T. Because of this, when a particular node is able to receive input from two sender nodes, it cannot simulate an event with any assurance that it would not be called upon to refrain from simulating the event, until it receives a message from both sender nodes. Waiting to receive a message from all inputs slows the simulation process down substantially and can easily result in a deadlock cycle where each node waits for a previous node, which amounts to the situation of a node waiting for itself.

To remedy the wait problem, artisans have been employing recovery and avoidance techniques. In the recovery technique, proposed by Chandy and Misra, upon detecting a deadlock, the processors in the network exchange messages in order to determine which of the waiting nodes can process their events in spite of the apparent deadlock. This discribed in K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," Communications of the ACM, Vol. 24, No. 4, April 1981, pp. 198–206. In the avoidance technique, on the other hand, certain types of nodes send null messages under specific conditions even when no instructions for other nodes are called for. By this technique, nodes can be advanced more quickly in their simulation time. Jefferson and Sowizral, in "Fast Concurrent Simulation Using the Time Warp Mechanism," Distributed Simulation, 1985, The Society for Computer Simulation Multiconference,

San Diego, Calif., suggest a different technique where each node is allowed to advance in its simulation time "at its own risk," but when a message arrives that would have caused some events to not have been simulated, then a "roll-back" is executed to undo the simulation that was done. Roll-back of a node may not be difficult, perhaps, but the fact that the simulated event(s) that need to be rolled back may have caused messages to be sent to other nodes does complicate the task substantially. To achieve the rollback, Jefferson et al. suggest the use of "anti-messages," which are messages that parallel the original messages, except that they cause the performance of some action that "undoes" the original action.

Neither of these techniques is very good because each potentially expands an inordinate amount of computation time in making sure that the overall simulation advances properly. The null message approach expends computing resources in generating, sending, and reading the null message; the recovery approach expends computing resources to detect and recover from a deadlock, and roll-back approach expends computing resources in simulating events and then undoing the work that was previously done.

SUMMARY OF THE INVENTION

Recognizing that in physical systems there is always some delay between the time when one part of the system does something, and time when another part of the system realizes that something was done, a simulation system is realized that avoids all blocking and advances the simulation time in an efficient manner. The efficiency is achieved by each node independently evaluating for itself a time interval that is not "at risk" and simulating all events that are within that evaluated time. A point in time is not "at risk" for a considered node if no other node in the system has a scheduled event that can affect the simulation at the considered node. By also restricting the simulation of scheduled events at any one time to a chosen simulated time segment (bounded lag) beginning with the lowest simulation time found among the nodes, allows the evaluation of the "at risk" time interval to be based on only a subset of the nodes that can potentially affect the simulation at the considered node. This simplification results from the the fact that there are delays between nodes, and that the lower bounds for those delays are fixed and known apriorily.

In simulating systems where some nodes affect other nodes only through intermediate nodes, opaque periods can be experienced when, because of the specific process that is being simulated, such an intermediate node "promises" that a particular route emanating from this node would be busy for a set period of time, and thereby also "promises" that no other node can use this route as a conduit to affect other nodes. That, in effect, increases the propagation delay from the nodes that use the busy intermediate route, which, in turn, increases the allowance for the simulation time within the bounded lag that is not "at risk".

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 illustrates the timing inter-relationships of events processed in a multi-processor environment with recognized delay between events and their effect on other events;

FIG. 2 depicts a block diagram of a multi-processor arrangement for simulating events in accordance with the principles of this invention;

FIG. 3 presents a flow chart describing the steps carried out in each of the processors of FIG. 2 in the course of event simulation; and

FIG. 4 describes one realization for the node controllers of FIG. 2.

DETAILED DESCRIPTION

As indicated above, one of the major problems with the prior art distributed simulation systems is their failure to realize and take advantage of the fact that a delay is always present between communicating physical subsystems. This invention takes advantage of this inherent delay, as described in detail below.

FIG. 1 presents a pictorial explanation that may aid in understanding the principles of this invention. Therein, vertical lines 21, 22, 23, 24, 25, 26 and 27 represent seven simulation nodes and their simulation time lines (where time advances upward). The circles along these lines (30–43) represent events that have been, or are scheduled to be processed (i.e., simulated). These events may cause change of value or state, i.e., other events, at the originating node or at some other nodes. For purposes of discussion, it is assumed that node 24 is the node of concern, but it is understood that the consideration undertaken by node 24 are concurrently taken by all other nodes.

The horizontal distances between line 24 and the other lines represent the time delays for events in other nodes to affect the value or state at node 24. Accordingly, event 30 processed at node 21 for time T_1 may cause an event40 at node 24 to be scheduled for some time not earlier than T_2, as shown in FIG. 1. The interval between T_1 and T_2 equals the delay between line 24 and line 21 (i.e., the horizontal distance between the lines).

The events depicted in FIG. 1 can be divided into two groups: those that have been simulated (and marked with a cross), and those that are yet to be simulated (un-crossed). The simulated events need not be considered for simulation because their effects are already represented by those events which have not been simulated (for example, event 30 has caused event 40; the first has been simulated, while the second is yet to be considered for processing).

Of the events that have yet to be simulated (33–43), event 33 in FIG. 1 is earliest in time from among all of the nodes. In this case, the time of event 33, forms the current floor simulation time of the system. The floor is depicted in FIG. 1 by dashed line 45. In accordance with the principles of this invention, a time interval beginning with the floor simulation time is selected for consideration. This time interval, which I call the bounded lag interval, can be a convenient time interval that takes into account the number of nodes in the system, the number of events to be simulated, and the computing power of the processors employed. All events scheduled within the bounded lag interval can be affected by events scheduled at other nodes within the bounded lag time interval, but only if those nodes are at a time distance from the affected node 24 that is less than the selected bounded lag interval. That reduces the number of nodes that need to be considered. In the FIG. 1 depiction, the bounded lag interval ends at dashed line 46; and as drawn, the nodes that need to be considered for their potential effect on node 24 are nodes 22, 23, 25 and 26. Nodes 21 and 27 are outside the bounded lag delay and their scheduled events within (or outside) the bounded lag need not be considered.

In considering the effects on node 24 in greater detail, one can take into account the time of the next scheduled event and reduce the number of nodes being considered even further. In the FIG. 1 embodiment, for instance, the next scheduled event is event 35, and as drawn, only nodes 23 and 25 need to be considered.

In determining whether event 35 is to be simulated, one can observe that only event 34 is scheduled early enough to have a possible impact on event 35. That event can be analyzed and if it is determined that it does not affect event 35, then event 35 can be simulated. Alternatively, it may prove even more efficient to refrain from processing event 35 simply because of the potential effect by event 34. In the following description, this alternative approach is taken because it saves the process of evaluating what event 34 may do.

FIG. 2 presents a block diagram of a concurrent event simulator. It comprises a plurality of node controllers 51 that are connected to a communications and common processing network 52. Network 52 performs the communication and synchronization functions, and in some applications it also performs some of the computations that are needed by the entire system. The node controllers can be realized with conventional stored program computers that may or may not be identical to each other. Each of the node controllers, which corresponds to a node in the FIG. 1 depiction, is charged with the responsibility to simulate a preassigned subsystem of the simulated system. Each controller C_i maintains an event list II_i that is executed by simulating each event in the list in strict adherence to the scheduled event simulation times. It should be remembered that the bounded lag interval is selectively fixed, and that in conformance with the selected bounded lag interval each controller 51 is cognizant of the processors with which it must interact to determine whether events are scheduled. The process by which the event simulations are enabled is carried out by the controllers in accordance with the above-described principles, as shown, for example, by the flowchart of FIG. 3.

The process described in FIG. 3 is carried out in iterations that encompass blocks 100 through 140. Block 100 is the flow control block. It tests whether the floor simulation time, T_{floor}, *is less than the end of the simulation time,* T_{end}. As longs as $T_{floor} < T_{end}$, the simulation continues by passing control to block 110. When T_{floor} reaches or exceeds T_{end}, the simulation ends. Block 110 determines the floor simulation time of the simulated system at each iteration. That is, block 110 determines the lowest event time of the scheduled events dispersed among the event lists (II_i) of controllers 51 (C_i) that are are yet to be simulated. Expressed mathematically, block 110 evaluates the floor simulation time in accordance with the equation

$$T_{floor} = \min_{1 \leq i \leq N} T_i$$

where N is the total number of node controllers 51 and T_i is the time of the event, e_i, which has the earliest scheduled time among the events e' in the event list II_i; i.e.,

$$T(e_i) = T_i = \min_{e' \in \pi_i} T(e').$$

Block 110 can be implemented in each of the controllers by having each controller broadcast to all other controllers (via network 52) its T_i and evaluate for itself the minimum T_i which defines T_{floor}. Alternatively, block 110 can be implemented within communications and common processing network 52 by having controllers 51 send their T_i values to network 52, and having network 52 select the minimum T_i and broadcast it as T_{floor} back to the controllers.

Having established the T_{floor}, and knowing the system's bounded lag interval, B, that limits the number of neighboring controllers with which a controller must communicate, in accordance with block 120, each of the controllers evaluates its earliest "at risk" time. This is accomplished by network 52 distributing the T_i information to neighboring controllers, as required, and each controller C_i evaluates the "at risk" demarcation point, α_i, from the T_i information. This "at risk" point is defined as the earliest time at which changes at the neighboring controllers can affect the history simulated by the controller, based on the neighboring controllers' own scheduled events or based on a response to an event from the controller itself (reflection). This is expressed by the following equation:

$$\alpha_i = \min_j [d(j,i) + \min\{T_j, d(i,j) + T_i\}].$$

Having determined the value of α_i which corresponds to the point in the simulated time beyond which the simulation of events at controller C_i is "at risk", in accordance with block 130, processor C_i simulates all or some of the scheduled events whose times are earlier that α_i. In block 140, the time T_i is advanced with each simulation of an event, and the simulated event is deleted from II_i. Concurrently, if in the course of simulating an event, new events are called to be scheduled, then those events are sent to network 52 for transmission to the appropriate node controllers. Similarly, if the execution of events is called to be blocked, that information is also sent to network 52, and thereafter to the appropriate node controllers for modification of the event lists.

The following carries out the example depicted in FIG. 1, assuming for the sake of simplifying the drawing that the situation remains static—i.e., none of the depicted events are cancelled and no unshown events are scheduled. After events 30, 31, and 32 have been simulated (denoted by the crossed circles) all of the node controllers communicate their earliest scheduled event times, T_i, to network 52 where T_{floor} is evaluated to correspond to dashed line 45. With reference to node 24, the bounded lag defined by the distance between dashed line 45 and dashed line 46 specifies that only nodes 22–26 need to be considered. In the course of that consideration, node 24 determines that scheduled event 34 at node 25 defines an "at risk" demarcation point 50. Since there are no events scheduled for node controller 24 between the time of T_{floor} and point 50, no progress in simulations is made by this controller. Concurrently, node controller 22 simulates event 33 (since it is positioned at T_{floor}, and no other event can affect it), and

node 25 simulates event 34 since neither node 26 nor node 24 (the closest nodes) have any events scheduled prior to the time of event 34. Node 27 probably also simulates event 36, but this is not certain, since FIG. 1 does not show all of the neighbors of node controller 27.

With events 33, 34, and 36 simulated and deleted from their respective event lists, the next iteration raises T_{floor} to the time of event 37 (and correspondingly raises the horizon or end of the bounded lag interval. This end is equal to $T_{floor}+B$). At this new level of T_{floor} the "at risk" demarcation point for node 24 is at point 49 (dictated by event 39 of node 23), and in accordance with this new "at risk" point, both events 35 and 40 are simulated within node controller 24. This completes the simulation of events scheduled for node 24 which are shown in FIG. 1. Concurrently at node 23, event 39 is simulated, event 37 at node 27 is simulated, but at node 26 event 38 is not simulated because it is beyond the "at risk" point of node 26 caused by the position of event 37 at node 27. The next T_{floor} moves to the time of event 38, and the process continues to simulate additional events.

The above description concentrates on evaluating the "at risk" demarcation point in connection with the direct effects of one node on another. In many physical systems, however, there are many instances where one subsystem affects another subsystem indirectly, i.e., through some other subsystem. This condition gives rise to the possibility that the intermediate node is either busy and unavailable generally, or is somehow sensitized to serve some nodes and not other nodes. Either situation can yield the condition that the delay from one node, A, to another node, C, through an intermediate node, B, is at times much longer than the sum of the of delays A to B and B to C. I call this additional delay an opaque period. Opaque periods have the potential for pushing forward the "at risk" demarcation point and, therefore, it is beneficial to account for this potential in evaluating α_i. Such accounting may be achieved by evaluating α_i iteratively as follows.

1: Set $\alpha_i^0 = +\infty$; $\beta_i^0 = T_i$; $k = 0$

2: synchronize

3: evaluate $\beta_i^{k+1} = \min_{\substack{j \in \text{neighbors}(i) \\ j \neq i}} \{d(j,i) + \beta_j^k\}$

4: evaluate $\alpha_i^{k+1} = \min\{\alpha_i^k, \beta_i^{k+1}\}$

5: synchronize

6: evaluate $A = \min_{1 \leq i \leq N} \beta_i^{k+1}$; broadcast value of A to all nodes

In the above, the term neighbors(i) refers to nodes that communicate directly with node i. The auxiliary variable β_i^k represents an estimate of the earliest time when events can affect node i after traversing exactly k links. It can be shown that the iteration test is always met within a relatively low number of steps, depending on the value of the bounded lag interval, B. To account for opaque periods, the evaluation of β is augmented to be

$$\beta_i^{k+1} = \min_{\substack{j \in \text{neighbors}(i) \\ j \neq i}} \{d(j,i) + \max\{\beta_j^k, op_{ji}\}\}$$

where op_{ji} is the end of opaque period (when communication unblocks) for node j in the direction of node i.

In some simulations it may turn out that the delays between subsystems are very small and that opaque periods predominate. In such systems the value of each α_i reduces to computing the minimum of the opaque periods relative to block i; to wit:

$$\alpha_i = \min_{\substack{j \in \text{neighbors}(i) \\ j \neq i}} op_{ji}$$

It may be observed that with each iteration the value of T_{floor} increases because the event determining that value can always be simulated. The movement of T_{floor} is affected, however, by how closely the nodes are separated and the scheduled events. One other observation that can be made is that the above-described procedure is very conservative, in the sense that an assumption is made that whenever an event is scheduled to be simulated at one node controller, it will always have an effect on the neighboring controller (after the appropriate delay). In physical systems, however, there are many situations where one part of a system performs many operations that affect none of its neighbor subsystems, or affect only one or very few of its neighbor subsystems. Knowledge that an event scheduled for simulation will not affect a neighboring node can be put to use to simulate more events concurrently (have fewer node controllers that are idle). This can be accomplished by communicating not only the T_i of the earliest scheduled event in each list, but also the effect that it may have on neighboring node controllers. If fact, each list can broadcast more than just the earliest scheduled event. The design decision that a practitioner must make, of course, is whether fewer iterations but more complex evaluations are economically justifiable.

FIG. 4 presents a block diagram of one realization for node controller 51. Although this embodiment relates to use of the iterative method for evaluating α (with the use of the auxiliary variable β) when the delays between changes in one subsystem and the effect of those changes on other subsystems are not insignificant, it will be appreciated by the skilled artisan that the other realizations for computing α_i are substantially similar to this realization. It will also be appreciated that although node controller 51 is shown in FIG. 2 as one of a plurality of controllers, such plurality can be realized within a single processor of sufficient computing power.

In FIG. 4, state register 56 defines the current state of the simulated subsystem and event list 53 specifies the events that are scheduled to be simulated. Processor 54 is the event simulation processor, and it is responsive to state register 56, to event list 53 and to register 58. Register 58 is the α register and, as the name implies, it stores the value of α for the controller. Based on the value of α and the scheduled time of the event at the top of the event list, processor 54 simulates the event in conformance with the state of the subsystem and develops a new state of the subsystem as well as, perhaps, new events. The new state is stored in register 56, new

events scheduled for the controller are stored in event list 53 via line 61, and events affecting other controllers are communicated out via line 62. Events produced at other controllers that may affect this controller are accepted via line 63 and stored in event list 53 through processor 54.

Whereas processor 54 is the event simulation processor, processor 55 is the synchronization processor. Processor 54 is shown in FIG. 4 as a separate processor but in practice a single processor may serve the function of both processor 54 and processor 55. Processor 55 receives information from event list 53 concerning the time of the event in list 53 that possesses the earliest simulation time. It transmits that information to other controllers via line 64 and receives like information from all other relevant nodes via line 65. From that information processor 55 develops the value of T_{floor} and stores that value in register 57. Processor 55 also receives β_i information from its neighbor controllers (controllers where changes can affect the controller directly) via line 66, and transmits its own β_i values via line 67. With the aid of T_{floor} and the other incoming information, processor 55 performs the iterative computations to develop the values of β_i^k and α_i^k. Those values are stored by processor 55 in registers 57 and 58, respectively.

It is to be understood that the foregoing descriptions are merely illustrative of my invention and that other implementations and embodiments which incorporate variations from the above may, nevertheless, incorporate the principles thereof. For example, the above assigns the task of computing T_{floor} to processor 55. However, it may be preferable to include computing means in the communication and common processing network of FIG. 2 where the T_{floor} is computed and distributed to the various node controllers.

I claim:

1. A method for simulating behavior of a system containing interacting subsystems, where a known minimum delay exists between changes occurring at one of said subsystems and the effects of said changes on others of said subsystems, said simulating being performed with a simulation system having a plurality of interconnected processors with each of said subsystems to be simulated being simulated on one of said processors, wherein the improvement comprises:

in each of said processors, a step of simulating events of subsystems, to be simulated in each of said processors, whose simulated time lies within a preselected interval following the earliest simulation time of events yet to be simulated by said processors.

2. The method of claim 1 wherein each of said steps of simulating is followed by a step of advancing said earliest simulation time of events yet to be simulated by said processors.

3. The method of claim 1 wherein said step of simulating includes a step of evaluating whether a scheduled event is to be simulated, based on changes occurring at a subset of said subsystems in response to previous simulation steps.

4. The method of claim 3 wherein the subsystems belonging to said subset are dependent on said preselected interval.

5. The method of claim 3 wherein said subset is controlled by said preselected interval and said delays between said subsystems.

6. The method of claim 4 wherein said subset includes the subsystems whose delays are within said preselected interval.

7. The method of claim 3 wherein said step of evaluating computes a simulated time value of α and said step of simulating simulates events whose simulated time is not greater than α.

8. The method of claim 7 wherein said α for subsystem i, α_i, is computed by

$$\alpha_i = \min_j [d(j,i) + \min\{T_j, d(i,j) + T_i\}].$$

where d(j,i) is said delay between changes occurring at subsystem j and their possible effect on system i,d(i,j) is said delay between changes occurring at subsystem i and their possible effect on system j, and T_j is the simulation time of subsystem j event to be simulated having the earliest simulation time from among all other subsystem j events to be simulated.

9. The method of claim 4 wherein said step of evaluating includes consideration of the condition that the effect of a change in one of said subsystems is blocked from propagating through another one of said subsystems.

10. The method of claim 3 wherein said step of evaluating computes a simulated time value of α and said step of simulating simulates events whose simulated time is not greater than α, where α for subsystem i, α_i, is the value of α_i^k in the final iteration that follows the steps:

1: Set $\alpha_i^0 = +\infty$; $\beta_i^0 = T_i$; $k = 0$

2: synchronize

3: evaluate $\beta_i^{k+1} = \min_{\substack{j \in \text{neighbors}(i) \\ j \neq i}} \{d(j,i) + \beta_j^k\}$

4: evaluate $\alpha_i^{k+1} = \min\{\alpha_i^k, \beta_i^{k+1}\}$

5: synchronize

6: evaluate $A = \min_{1 \leq i \leq N} \beta_i^{k+1}$; broadcast value of A to all nodes

where T_i is the time of the event in subsystem i having the earliest simulation time, β_i^k is an auxiliary variable value at iteration k, and d(j,i) is said delay between changes occurring at subsystem j and their possible effect on subsystem i.

11. The method of claim 3 wherein said step of evaluating computes a simulated time value of α and said step of simulating simulates events whose simulated time is not greater than α, where α for subsystem i, α_i, is the value of α_i^k in the final iteration that follows the steps:

1: Set $\alpha_i^0 = +\infty$; $\beta_i^0 = T_i$; $k = 0$

2: synchronize

-continued

3: evaluate $\beta_i^{k+1} \doteq \min_{\substack{j \in neighbors(i) \\ j \neq i}} \{d(j,i) + \max\{\beta_j^k, op_{ji}\}\}$

4: evaluate $\alpha_i^{k+1} = \min\{\alpha_i^k, \beta_i^{k+1}\}$

5: synchronize

6: evaluate $A =$

$$\min_{1 \leq i \leq N} \beta_i^{k+1}$$; broadcast value of A to all nodes

where T_i is the time of the event in subsystem i having the earliest simulation time, β_i^k is an auxiliary variable value at iteration k,d(j,i) is said delay between changes occurring at subsystem j and their possible effect on subsystem i, and op_{ji} is the opaque period for subsystem j in the direction of subsystem i.

12. The method of claim 4 wherein said step of evaluating computes a value of α and said step of simulating simulates events whose simulated time is not greater than α, where α for subsystem i, α_i, is equal to

$$\min_{\substack{j \in neighbors(i) \\ j \neq i}} op_{ji}$$

where op_{ij} is the opaque period for subsystem j in the direction of subsystem i.

13. A system for performing discrete event simulation of a system having a plurality of interacting subsystems, comprising:

a plurality of blocks, each block simulating a preselected subsystem and including an event list associated with said preselected subsystem and means for maintaining information concerning the state of said preselected subsystem, and α, where α is an estimate of the earliest simulated time for which modification parameters can be modified by neighboring subsystems, where said modification parameters include the information in said associated state registers and the events in said associated event; and

a controller system for repetitively performing simulation of events in said event lists and re-evaluating the values of said α and of T_{floor}, where T_{floor} is the lowest event time of scheduled events to be simulated by said plurality of blocks.

14. The system of claim 14 wherein said controller system comprises

a plurality of interconnected controllers communicating with said blocks, with each controller repetitively performing simulation of events in said event lists and re-evaluating the value of said α.

15. The system of claim 14 wherein said controller system comprises

an event evaluation processor associated with each of said blocks, connected to its associated event list, its associated means for maintaining state information, and to neighboring blocks, which are blocks that simulate subsystems that directly affect, or can be affected by, said preselected subsystem, where said event processor performs simulation of events in said associated event list and modifies said associated event list and the event lists of said neighboring blocks, as required by said simulation of events;

a synchronization processor associated with each of said blocks, connected to its associated event list and means for maintaining α, for developing values of said α; and

means for interconnecting said plurality of event evaluation processors and synchronization processors.

16. The system of claim 16 wherein said synchronization processor develops a value of α based on values of T_j, from a preselected subset of said subsystems, where T_j is the simulation time of the event in subsystem j having the earliest simulation time.

17. The system of claim 16 where each of said synchronization processors develops a value of α for its block based on a consideration of potential changes in neighbors of its block.

18. The system of claim 17 wherein said consideration is iterative.

19. The system of claim 18 wherein said consideration takes into account opaque periods.

20. The system of claim 16 where each of said synchronization processors develops a value of α for its block based on a consideration of opaque periods of its neighbors.

21. The system of claim 14 wherein said controller system comprises

a plurality of controllers communicating with said blocks, with each controller repetitively performing simulation of events in said event lists and re-evaluating the value of said α; and

means for interconnecting said controllers.

22. The system of claim 21 wherein said means for interconnecting receives a T_i from each of said event lists, where T_i is the simulated time of the event to be simulated in said event list having the earliest simulation time, develops T_{floor} which is min (T_i), and returns T_{floor} to each of said controllers.

23. A system for performing discrete event simulation comprising:

a plurality of blocks equal in number to the number of simulated interacting subsystems, each block simulating a preselected subsystem and including an event list associated with said preselected subsystem and registers for storing information concerning T_{floor}, α, β and state information of said preselected subsystem;

means for interconnecting said plurality of blocks;

an event processing controller associated with each of said blocks, connected to its associated event list, its associated state registers, and to neighboring blocks, which are blocks that simulate subsystems that directly affect, or can be affected by, said preselected subsystem, where said event controller performs simulation of events in said associated event list and modifies said associated event list and the event lists of said neighboring blocks, as required by said simulation of events; and

a synchronization controller connected to its associated event list, T_{floor}, α, and β registers, and also connected to said means for interconnecting, for developing values for said T_{floor}, α, and β registers, where T_{floor} is the earliest simulated time of an event to be simulated found in any of said event lists, α is an estimate of the earliest simulated time for which the information in said associated state registers or the events in said associated event list

can be modified by neighboring subsystems, and β is an auxiliary estimate employed in evaluating α.

24. A system for performing discrete event simulation comprising:

a plurality of blocks equal in number to the number of simulated interacting subsystems, each block simulating a preselected subsystem and including an event list associated with said preselected subsystem and registers for storing information concerning T_{floor}, α, and state information of said preselected subsystem;

means for interconnecting said plurality of blocks;

an event processing controller associated with each of said blocks, connected to its associated event list, its associated state registers, and to a subset of said blocks, where said event controller performs simulation of events in said associated event list and modifies said associated event list and the event lists of said neighboring blocks, as required by said simulation of events; and

a synchronization controller connected to its associated event list, T_{floor} and, α, registers, and also connected to said means for interconnecting, for developing values for said T_{floor}, and α, registers, where T_{floor} is the earliest simulated time of an event to be simulated found in any of said event lists, and α is an estimate of the earliest simulated time for which the information in said associated state registers or the events in said associated event list can be modified by neighboring subsystems.

* * * * *